

# Stocks Near-Real-Time Data Analysis and Visualization Project

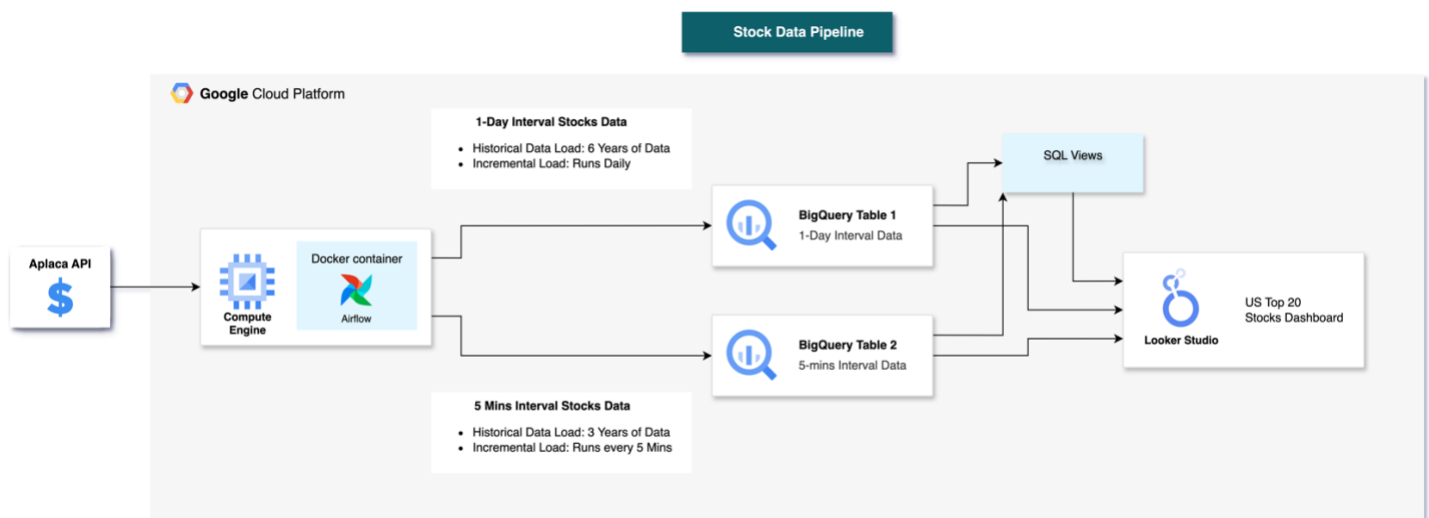
## Overview

In this project, I am running ETL pipelines using Apache Airflow to collect (6 years of historical + near real-time) stock data from the Alpaca API, process it, and store it in two Google BigQuery tables:

- Table 1: Stores 1-day interval stock data (One-time ETL run + incremental ETL runs daily).
- Table 2: Stores 5-minute interval stock data (One-time ETL run + incremental ETL runs every 10 minutes).

I am using Google Cloud Engine to run Airflow in a Docker container spun up using a docker-compose file. I have connected Looker Studio to the BigQuery tables and SQL views to create different visualizations for analyzing stocks.

## Architecture:



## Technology Stack

- **Programming Language:** Python, SQL
- **APIs (Data Source):** Alpaca
- **Containerization:** Docker, Docker Compose
- **Database:** Google BigQuery
- **Orchestrator:** Apache Airflow
- **Data Visualization:** Looker Studio
- **Cloud Platform:** Google Cloud Platform (GCP)
- **Development Environment:** Visual Studio Code (VS Code) with "Remote Development" Extension

### Aplaca API details:

- **Key:** PKAFPE8IL5495B6P4106
- **Secret:** yde2oINHNXvDSUpsJUKART2pWJ7sMeXEbz2Oe4vp

### Alpha Vantage API details:

- **API key:** M5DQ8BDRKJOVFU5I

Sample output of Aplaca API : <https://data.alpaca.markets/v2/stocks/AAPL/bars>

```
{
  'bars': [
    {
      'c': 216.24,
      'h': 216.78,
      'l': 211.97,
      'n': 589469,
      'o': 212.1,
      't': '2024-08-09T04:00:00Z',
      'v': 42201646,
      'vw': 215.199872
    },
  ],
  'next_page_token': None,
  'symbol': 'AAPL'
}
```

**c (Close Price):** The closing price of the stock on that day.

**h (High Price):** The highest price reached during the trading day.

**l (Low Price):** The lowest price reached during the trading day.

**n (Number of Trades):** The number of trades executed on that day.

**o (Open Price):** The opening price of the stock for the day.

**t (Timestamp):** The date and time of the data point, in ISO 8601 format (UTC).

**v (Volume):** The total number of shares traded on that day.

**vw (Volume Weighted Average Price):** The volume-weighted average price for the day.

## Infrastructure setup Process:

1. Create Google Cloud Engine instance: **minimum n2-standard-2** type (Allow ports 8080,8081,22,http)
2. **Setting Up Docker on Google Cloud Engine (Debian)**

### Install Docker:

- `sudo apt-get update`
- `sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg`
- `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
- `echo "deb [arch=amd64] https://download.docker.com/linux/debian bookworm stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`
- `sudo apt-get update`
- `sudo apt-get install -y docker-ce`
- `sudo systemctl status docker`
- `sudo usermod -aG docker $USER`
- `newgrp docker`
- `docker run hello-world`
- `docker --version`

### Install Docker Compose:

- `sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`
- `sudo chmod +x /usr/local/bin/docker-compose`
- `docker-compose --version`

3. **Run Airflow on GCE using docker compose. We would need postgres for airflow metadata to be stored.**

- Create a docker-compose.yml File:

version: "3.8"

services:

postgres:

image: postgres:latest

container\_name: postgres

environment:

POSTGRES\_DB: airflow

POSTGRES\_USER: airflow  
POSTGRES\_PASSWORD: airflow

ports:

- "5432:5432"

volumes:

- postgres\_data:/var/lib/postgresql/data

networks:

- my\_network

airflow-init:

image: apache/airflow:latest

container\_name: airflow\_init

environment:

AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'False'

AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN:

postgresql+psycopg2://airflow:airflow@postgres:5432/airflow

entrypoint: /bin/bash -c "airflow db init"

networks:

- my\_network

airflow-webserver:

image: apache/airflow:latest

container\_name: airflow\_webserver

environment:

AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'False'

AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN:

postgresql+psycopg2://airflow:airflow@postgres:5432/airflow

ports:

- "8081:8080"

volumes:

- airflow\_dags:/opt/airflow/dags

- airflow\_logs:/opt/airflow/logs

- airflow\_plugins:/opt/airflow/plugins

command: ["airflow", "webserver"]

networks:

- my\_network

depends\_on:

- postgres

airflow-scheduler:

image: apache/airflow:latest

container\_name: airflow\_scheduler

```

environment:
  AIRFLOW__CORE__EXECUTOR: LocalExecutor
  AIRFLOW__CORE__LOAD_EXAMPLES: 'False'
  AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
volumes:
  - airflow_dags:/opt/airflow/dags
  - airflow_logs:/opt/airflow/logs
  - airflow_plugins:/opt/airflow/plugins
command: ["airflow", "scheduler"]
networks:
  - my_network
depends_on:
  - postgres

```

```

volumes:
  postgres_data:
  airflow_dags:
  airflow_logs:
  airflow_plugins:

```

```

networks:
  my_network:
    driver: bridge

```

### Run Docker Compose:

- `docker-compose up -d`
- `docker-compose up --build`

### Verify Running Containers:

- `docker ps`

### 4. Create 2 tables on BigQuery to store the data:

- **historical\_stock\_prices:** Store 1-day interval stock data
- **realtime\_stock\_prices:** Store 5-min interval stock data

```

CREATE TABLE stocksdatacsv-433003.stocks_data.historical_stock_prices (
  symbol STRING,
  close_price FLOAT64,
  high_price FLOAT64,

```

```

        low_price FLOAT64,
        number_of_trades INT64,
        open_price FLOAT64,
        timestamp TIMESTAMP,
        volume INT64,
        volume_weighted_average_price FLOAT64
    )

CREATE TABLE stocksdatacsv-433003.stocks_data.realtime_stock_prices (
    symbol STRING,
    close_price FLOAT64,
    high_price FLOAT64,
    low_price FLOAT64,
    number_of_trades INT64,
    open_price FLOAT64,
    timestamp TIMESTAMP,
    volume INT64,
    volume_weighted_average_price FLOAT64
)

```

5. **Access Airflow UI:** [http://<PUBLIC\\_IP\\_of\\_GCE\\_instance>:8081/](http://<PUBLIC_IP_of_GCE_instance>:8081/)

6. **To login to Airflow, you need to create a user in Airflow:**

```

docker exec -it airflow_webserver airflow users create \
--username atul \
--firstname Atul \
--lastname Nayak \
--role Admin \
--email abc@northeastern.edu \
--password xxxxxx

```

7. **Access Airflow Container Terminal:**

- `docker exec -it <container ID> /bin/bash`
- `docker exec -u root -it <container ID> /bin/bash` (For root access)

8. **Create DAG Files in Airflow here:** `/opt/airflow/dags`

- Install Nano Editor (Optional):
  - `apt-get update`
  - `apt-get install -y nano`

- Create Dag file using nano in dags directory: `nano pull_historic_data_dag.py`

## 9. Access the Dag code on airflow running in GCE using VS code:

- Download “Remote Development” extensions on VS code
- Check from Mac CLI first, if you can access the GCE using : `ssh nayakatul1k999@<public_IP>`
- If it fails:
  - **On your Mac CLI, navigate to the SSH directory:** `cd ~/.ssh`
  - **Display your public key:** `cat id_rsa.pub`
  - **Copy the output and paste it into the GCE CLI:** `nano ~/.ssh/authorized_keys`

### Update Permissions on GCE:

- `chmod 600 ~/.ssh/authorized_keys`
- `chmod 700 ~/.ssh`

### Adjust File Ownership and Permissions on GCE instance:

- `chown nayakatul1k999:nayakatul1k999 ~/.ssh/authorized_keys`
- `chown nayakatul1k999:nayakatul1k999 ~/.ssh/config`
- `chown nayakatul1k999:nayakatul1k999 ~/.ssh/known_hosts`
- `chmod 600 ~/.ssh/authorized_keys`
- `chmod 600 ~/.ssh/config`
- `chmod 644 ~/.ssh/known_hosts`

### Set Up Docker Context if required:

- `docker context create some-context-label --docker "host=ssh://nayakatul1k999@<public_IP>"`
- `docker context use some-context-label`
- Try connecting from MAC CLI now using: `ssh nayakatul1k999@<public_IP>`
- If it is a success, it means permission is proper to connect to GCE from Mac local. Now, you can try connecting from VS code:
  - Go to >< symbol > Search “connect to host” > Add new SSH host (if not already present)
  - On MAC CLI:
    - `cd ~/.ssh`
    - `nano config` ( enter GCE details like Public Ip, username etc)

```
Host stockvm
  HostName 35.192.119.86
  User nayakatul1k999
```

IdentityFile ~/.ssh/id\_rsa

- **Once you have access to dag file through VS code, enter and run the dag python scripts**
  - To Save the dag scripts using VScode, you must change permission of the dag file through root airflow account in container:
    - `chown root:root /opt/airflow/dags/pull_historic_stock_data_dag.py`
    - `chmod 666 /opt/airflow/dags/pull_historic_stock_data_dag.py`
- **Make sure to create a service account on google cloud and download the service account file and store it in the airflow container (in the path: /opt/airflow/dags/ ), so that it can authenticate.**
  - You can use scp command to transfer the file from mac local to airflow container in GCE. Run this on MacBook CLI:

```
scp ./stocksdatacsv-433003-f8dc42626436.json  
nayakatul1k999@34.172.94.36:/home/nayakatul1k999/
```

- To transfer the file from GCE local to Airflow container, run this on GCE CLI:

```
docker cp /path/to/local/stocksdatacsv-433003-f8dc42626436.json  
<container_id>:/opt/airflow/dags/stocksdatacsv-433003-f8dc42626436.json
```

- **To run the dag file on Airflow:**
  - `Python dag_file_name.py`
  - `airflow dags list`
  - `airflow dags trigger dag_name`
  - `airflow dags test <dag_id>` (for troubleshooting)
- **After this, you must see the dag running on Airflow UI.**
- **Run all the dags → Run “One-time-run” dags first**

## 10. Check if data is present in BigQuery

## 11. Create SQL views for Looker Visualizations:

- **SQL Views for both “historical” table to change time to EST and map stock symbol to real names:**

```
CREATE OR REPLACE VIEW `stocksdatacsv-  
433003.stocks_data.historical_stock_prices_est` AS  
SELECT  
symbol,
```



```

CASE
  WHEN symbol = 'AAPL' THEN 'Apple Inc.'
  WHEN symbol = 'MSFT' THEN 'Microsoft Corp.'
  WHEN symbol = 'GOOGL' THEN 'Alphabet Inc.'
  WHEN symbol = 'AMZN' THEN 'Amazon.com Inc.'
  WHEN symbol = 'TSLA' THEN 'Tesla Inc.'
  WHEN symbol = 'META' THEN 'Meta Platforms Inc.'
  WHEN symbol = 'NVDA' THEN 'NVIDIA Corp.'
  WHEN symbol = 'BRK.B' THEN 'Berkshire Hathaway Inc.'
  WHEN symbol = 'JPM' THEN 'JPMorgan Chase & Co.'
  WHEN symbol = 'V' THEN 'Visa Inc.'
  WHEN symbol = 'MA' THEN 'Mastercard Inc.'
  WHEN symbol = 'WMT' THEN 'Walmart Inc.'
  WHEN symbol = 'DIS' THEN 'The Walt Disney Co.'
  WHEN symbol = 'HD' THEN 'The Home Depot Inc.'
  WHEN symbol = 'NFLX' THEN 'Netflix Inc.'
  WHEN symbol = 'PYPL' THEN 'PayPal Holdings Inc.'
  WHEN symbol = 'INTC' THEN 'Intel Corp.'
  WHEN symbol = 'CSCO' THEN 'Cisco Systems Inc.'
  WHEN symbol = 'ADBE' THEN 'Adobe Inc.'
  WHEN symbol = 'ORCL' THEN 'Oracle Corp.'
END AS company_name,
close_price,
high_price,
low_price,
number_of_trades,
open_price,
DATETIME(TIMESTAMP(timestamp), 'America/New_York') AS et_datetime,
volume,
volume_weighted_average_price
FROM
  `stocksdatscsv-433003.stocks_data.realtime_stock_prices`

```

- **SQL View for both “realtime” table to change time to EST and map stock symbol to real names:**

```

CREATE OR REPLACE VIEW `stocksdatscsv-
433003.stocks_data.realtime_stock_prices_est` AS
SELECT
  symbol,

```

```

CASE
  WHEN symbol = 'AAPL' THEN 'Apple Inc.'
  WHEN symbol = 'MSFT' THEN 'Microsoft Corp.'
  WHEN symbol = 'GOOGL' THEN 'Alphabet Inc.'
  WHEN symbol = 'AMZN' THEN 'Amazon.com Inc.'
  WHEN symbol = 'TSLA' THEN 'Tesla Inc.'
  WHEN symbol = 'META' THEN 'Meta Platforms Inc.'
  WHEN symbol = 'NVDA' THEN 'NVIDIA Corp.'
  WHEN symbol = 'BRK.B' THEN 'Berkshire Hathaway Inc.'
  WHEN symbol = 'JPM' THEN 'JPMorgan Chase & Co.'
  WHEN symbol = 'V' THEN 'Visa Inc.'
  WHEN symbol = 'MA' THEN 'Mastercard Inc.'
  WHEN symbol = 'WMT' THEN 'Walmart Inc.'
  WHEN symbol = 'DIS' THEN 'The Walt Disney Co.'
  WHEN symbol = 'HD' THEN 'The Home Depot Inc.'
  WHEN symbol = 'NFLX' THEN 'Netflix Inc.'
  WHEN symbol = 'PYPL' THEN 'PayPal Holdings Inc.'
  WHEN symbol = 'INTC' THEN 'Intel Corp.'
  WHEN symbol = 'CSCO' THEN 'Cisco Systems Inc.'
  WHEN symbol = 'ADBE' THEN 'Adobe Inc.'
  WHEN symbol = 'ORCL' THEN 'Oracle Corp.'
END AS company_name,
close_price,
high_price,
low_price,
number_of_trades,
open_price,
DATETIME(TIMESTAMP(timestamp), 'America/New_York') AS et_datetime,
volume,
volume_weighted_average_price
FROM
  `stocksdatabcsv-433003.stocks_data.realtime_stock_prices`

```

- **SQL View to get latest price of stocks for scorecard on Looker:**

```

CREATE OR REPLACE VIEW `stocksdatabcsv-433003.stocks_data.latest_stock_prices`
AS
WITH latest_prices AS (
  SELECT

```

```

        hsp.company_name,
        hsp.close_price,
        hsp.et_datetime
FROM
    `stocksdatacsv-433003.stocks_data.realtime_stock_prices_est` AS hsp
JOIN
    (
        SELECT
            company_name,
            MAX(et_datetime) AS max_timestamp
        FROM
            `stocksdatacsv-433003.stocks_data.realtime_stock_prices_est`
        GROUP BY
            company_name
        ) AS latest
ON
    hsp.company_name = latest.company_name
    AND hsp.et_datetime = latest.max_timestamp
),
historical_prices AS (
    SELECT
        hsp.company_name,
        hsp.close_price,
        hsp.et_datetime,
        ROW_NUMBER() OVER (PARTITION BY hsp.company_name ORDER BY
hsp.et_datetime DESC) AS rn
    FROM
        `stocksdatacsv-433003.stocks_data.historical_stock_prices_est` hsp
JOIN
    latest_prices lsp
ON
    hsp.company_name = lsp.company_name
WHERE
    hsp.et_datetime < lsp.et_datetime
)
SELECT
    lsp.company_name,
    lsp.close_price AS latest_close_price,
    hst.close_price AS previous_close_price
FROM
    latest_prices lsp
LEFT JOIN
    historical_prices hst
ON

```

```
lsp.company_name = hst.company_name  
AND hst.rn = 1;
```

**NOTE:** On Looker, to generate the scorecard (that shows the latest stock price above the graph, I connect the **latest\_stock\_prices** view and set **latest\_close\_price** in metric and **previous\_close\_price** in comparison metric.

- **For a toggle to filter values for 1D,5D,1M,6M,1Y,3Y on Looker, we need to add a parameter for :**
  - On Looker Studio > data source > Add a New Parameter:
  - Name the parameter as like "Period"
  - Set the Parameter Type to Text.
  - Provide possible values such as "1D," "5D," , "15D", "1M", "6M", "1Y", "3Y"
  - **To find Custom Query section :** Resource > Manage Added Data Source > Select the table > Edit connection > Custom Query
  - Enter this Custom SQL query there:

```
WITH data_source AS (  
  SELECT  
    symbol,  
      company_name,  
    close_price,  
    high_price,  
    low_price,  
    number_of_trades,  
    open_price,  
    et_datetime,  
    volume,  
    volume_weighted_average_price,  
    'table_1' AS table_source  
  FROM  
    `stocksdatacsv-433003.stocks_data.realtime_stock_prices_est`  
  WHERE  
    @period IN ('1D', '5D','15D')  
  
  UNION ALL  
  
  SELECT  
    symbol,  
      company_name,
```

```

        close_price,
        high_price,
        low_price,
        number_of_trades,
        open_price,
        et_datetime,
        volume,
        volume_weighted_average_price,
        'table_2' AS table_source
FROM
    `stocksdatacsv-433003.stocks_data.historical_stock_prices_est`
WHERE
    @period IN ('1M','6M', '1Y', '3Y', '6Y')
)

SELECT
    symbol,
    company_name,
    close_price,
    high_price,
    low_price,
    number_of_trades,
    open_price,
    et_datetime,
    volume,
    volume_weighted_average_price
FROM
    data_source
WHERE
    et_datetime BETWEEN
    CASE
        WHEN @period = '1D' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 DAY)
        WHEN @period = '5D' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 5 DAY)
        WHEN @period = '1M' THEN DATETIME_SUB((SELECT MAX(et_datetime)
FROM data_source), INTERVAL 15 DAY)
        WHEN @period = '1M' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 MONTH)
        WHEN @period = '6M' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 6 MONTH)
        WHEN @period = '1Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 YEAR)

```

```

        WHEN @period = '3Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 3 YEAR)
        WHEN @period = '6Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 6 YEAR)
        ELSE DATETIME_SUB((SELECT MAX(et_datetime) FROM data_source), INTERVAL 1
DAY) -- Default fallback
    END
    AND CURRENT_DATETIME()

```

- **Activate Necessary Parameters:** @DS\_START\_DATE and @DS\_END\_DATE parameters below the query editor.

**NOTE:** The above custom query uses “**realtime\_stock\_prices\_est**” prices table for parameter ‘1D’, ‘5D’, ‘15D’ and uses “**historical\_stock\_prices\_est**” table for parameters ‘1m’, ‘6m’, ‘1Y’, ‘3Y’ based on the parameter entered by the user.

- **Custom Data Source code for latest price scorecard just above the Linechart:**

```

WITH data_source AS (
    SELECT
        symbol,
        company_name,
        close_price,
        high_price,
        low_price,
        number_of_trades,
        open_price,
        et_datetime,
        volume,
        volume_weighted_average_price,
        'table_1' AS table_source
    FROM
        `stocksdatacsv-433003.stocks_data.realtime_stock_prices_est`
    WHERE
        @period IN ('1D', '5D', '15D')

    UNION ALL

    SELECT
        symbol,
        company_name,
        close_price,
        high_price,

```

```

        low_price,
        number_of_trades,
        open_price,
        et_datetime,
        volume,
        volume_weighted_average_price,
        'table_2' AS table_source
FROM
    `stocksdatacsv-433003.stocks_data.historical_stock_prices_est`
WHERE
    @period IN ('1M', '6M', '1Y', '3Y', '6Y')
),

```

```

timestamped_data AS (
    SELECT
        symbol,
        company_name,
        close_price,
        high_price,
        low_price,
        number_of_trades,
        open_price,
        et_datetime,
        volume,
        volume_weighted_average_price,
        MIN(et_datetime) OVER (PARTITION BY symbol) AS min_et_datetime,
        MAX(et_datetime) OVER (PARTITION BY symbol) AS max_et_datetime
    FROM
        data_source
    WHERE
        et_datetime BETWEEN
        CASE
            WHEN @period = '1D' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 DAY)
            WHEN @period = '5D' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 5 DAY)
            WHEN @period = '15D' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 15 DAY)
            WHEN @period = '1M' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 MONTH)
            WHEN @period = '6M' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 6 MONTH)
            WHEN @period = '1Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 1 YEAR)

```

```

        WHEN @period = '3Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 3 YEAR)
        WHEN @period = '6Y' THEN DATETIME_SUB((SELECT MAX(et_datetime) FROM
data_source), INTERVAL 6 YEAR)
        ELSE DATETIME_SUB((SELECT MAX(et_datetime) FROM data_source), INTERVAL 1
DAY) -- Default fallback
    END
    AND CURRENT_DATETIME()
),

```

```

max_realtime_timestamps AS (
    SELECT
        symbol,
        MAX(et_datetime) AS max_et_datetime
    FROM
        `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est`

    GROUP BY
        symbol
),

```

```

latest_realtime_data AS (
    SELECT
        r.symbol,
        r.close_price AS max_close_price
    FROM
        max_realtime_timestamps mrt
    JOIN
        `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est` r
    ON
        mrt.symbol = r.symbol
        AND mrt.max_et_datetime = r.et_datetime
)

```

```

SELECT
    ts.symbol,
    ts.company_name,
    MAX(CASE WHEN ts.et_datetime = ts.min_et_datetime THEN ts.close_price END) AS
min_close_price,
    lr.max_close_price
FROM
    timestamped_data ts
LEFT JOIN
    latest_realtime_data lr

```



```

ON
    ts.symbol = lr.symbol
GROUP BY
    ts.symbol,
    ts.company_name,
    lr.max_close_price
ORDER BY
    ts.symbol;

```

**NOTE:** Here I have used a custom SQL query as a data source for the latest price of the stock based on the parameter user enters (1D or 1M...). The above custom query uses “**realtime\_stock\_prices\_est**” prices table for parameter ‘1D’, ‘5D’, ‘15D’ and uses “**historical\_stock\_prices\_est**” table for parameters ‘1m’, ‘6m’, ‘1Y’, ‘3Y’ based on the parameter entered by the user. But for latest close price I am always using realtime table.

- **SQL view for top gainer, loser stocks table on Looker:**

```

CREATE OR REPLACE VIEW `stocksdatscsv-433003.stocks_data.top_gain_loser_stocks`
AS
select company_name, latest_close_price, previous_close_price,
case when(latest_close_price-previous_close_price) < 0 then (latest_close_price-
previous_close_price) end as top_losers,
case when(latest_close_price-previous_close_price) > 0 then (latest_close_price-
previous_close_price) end as top_gainers,
case when (latest_close_price-previous_close_price) < 0 then (latest_close_price-
previous_close_price)/previous_close_price*100 end as percent_loss,
case when (latest_close_price-previous_close_price) > 0 then (latest_close_price-
previous_close_price)/previous_close_price*100 end as percent_gain
from `stocksdatscsv-433003.stocks_data.latest_stock_prices`

```

- **SQL view for day range for every stock:**

```

CREATE OR REPLACE VIEW `stocksdatscsv-433003.stocks_data.day_high_low_range` AS
select company_name, Max(high_price) as max_day_price, Min(low_price) as
min_day_price from `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est`
where date(et_datetime) = (select max(date(et_datetime))
from `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est`)
group by company_name

```

- **SQL view for 52 week high and low:**

```

CREATE OR REPLACE VIEW `stocksdatscsv-
433003.stocks_data.fifty_two_week_high_low` AS
SELECT company_name, MAX(high_price) AS max_52_week_price, MIN(low_price) AS
min_52_week_price
FROM `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est`
WHERE date(et_datetime) BETWEEN DATE_SUB((SELECT MAX(date(et_datetime))
FROM `stocksdatscsv-
433003.stocks_data.realtime_stock_prices_est`), INTERVAL 52 WEEK)
AND (SELECT MAX(date(et_datetime))
FROM `stocksdatscsv-
433003.stocks_data.realtime_stock_prices_est`)
GROUP BY
company_name;

```

- **SQL view for latest volume for each stock:**

```

CREATE OR REPLACE VIEW `stocksdatscsv-433003.stocks_data.latest_stock_volumes`
AS
SELECT hsp.company_name, hsp.et_datetime, hsp.volume,
hsp.volume_weighted_average_price
FROM `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est` AS hsp
JOIN (SELECT company_name, MAX(et_datetime) AS max_timestamp
FROM `stocksdatscsv-433003.stocks_data.realtime_stock_prices_est`
GROUP BY company_name) AS latest ON hsp.company_name = latest.company_name
AND hsp.et_datetime = latest.max_timestamp

```

## Looker Visualization:

1. Set the theme → Size 1800 x 2300

2. Displaying stock trend graph:

- Used Community visualization **line chart (by ClickInsight)**
- Connected the line chart to **realtime\_stock\_prices\_est** view
- Used **“Close\_price”** as metric
- To format the date for the Community visualization line chart, I created a **calculated field** and named it **“et\_1”** and used this formula:

```

FORMAT_DATETIME('%Y-%m-%d %H:%M', et_datetime)

```

- Used **“et\_1”** in Axis dimension
- Used **“et\_1”** in Sort and set it to ascending (for time in ascending order)

### 3. For filtering the stocks names:

- Connected to **realtime\_stock\_prices\_est** view.
- Used **“drop down list”** control
- Used **“company\_name”** as the control field.

### 4. To show the latest stock price for above the graph:

- Used scorecard chart
- Connected a **custom data source**.
- I created a new parameter called **“Period1”** but assigned the parameter ID of Period because I wanted use on parameter filter.
- Set **max\_close\_price** in metric and **min\_close\_price** in comparison metric.

### 5. To Filter trend based on "1D," "5D," "1M", "6M", "1Y", "3Y" :

- Used **“fixed-size list”** control
- Added **“period”** parameter that we created in the control field.

### 6. Top Gainer table:

- Used **Table Chart**
- Connected **top\_gain\_loser\_stocks** view
- **“company\_name”** in Dimension
- **“latest\_close\_price”, “previous\_close\_price”, “top\_gainers”, “percent\_gain”** in Metric
- Sort descending based on **“percent\_gain”**

### 7. Top Gainer table:

- Used **Table Chart**
- Connected **top\_gain\_loser\_stocks** view
- **“company\_name”** in Dimension
- **“latest\_close\_price”, “previous\_close\_price”, “top\_losers”, “percent\_loss”** in Metric
- Sort ascending based on **“percent\_loss”**

### 8. Day Range → Which is next to Line Chart:

- Used **2 Scorecard chart**, one for Min and another for Max
- Connected to **day\_high\_low\_range** view
- Set “**min\_day\_price**” in Metric for Min Score card
- Set “**max\_day\_price**” in Metric for Max Score card
- Keep the scorecards next to each other

#### 9. 52 Week Range → Which is next to Line Chart:

- Used **2 Scorecard chart**, one for Min and another for Max
- Connected to **fifty\_two\_week\_high\_low** view
- Set “**min\_52\_week\_price**” in Metric for Min Score card
- Set “**max\_52\_week\_price**” in Metric for Max Score card
- Keep the scorecards next to each other

#### 10. Volume → Which is next to Line Chart:

- Used **a Scorecard chart**
- Connected to **latest\_stock\_volumes** view
- Set “**volume**” in Metric
- Set “**et\_datetime**” in Date Range Dimension

#### Historic data load, Python dag to pull data with 1 day interval (Run once)

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta, timezone
import requests
from google.cloud import bigquery
from google.oauth2 import service_account
import pytz

# Configuration
ALPACA_API_KEY = 'PKAFPE8IL5495B6P4106'
ALPACA_API_SECRET = 'yde2oINHNXvDSUpsJUKART2pWJ7sMeXEbz2Oe4vp'
BASE_URL = 'https://data.alpaca.markets'
HEADERS = {'APCA-API-KEY-ID': ALPACA_API_KEY, 'APCA-API-SECRET-KEY': ALPACA_API_SECRET}
BIGQUERY_PROJECT_ID = 'stocksdatacsv-433003'
BIGQUERY_DATASET = 'stocks_data'
```

```
BIGQUERY_TABLE = 'historical_stock_prices'
SERVICE_ACCOUNT_FILE = '/opt/airflow/dags/stocksdatacsv-433003-f8dc42626436.json'
```

```
# List of top 20 US stocks
```

```
TOP_20_STOCKS = [
    'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'META', 'NVDA', 'BRK.B', 'JPM', 'V',
    'MA', 'WMT', 'DIS', 'HD', 'NFLX', 'PYPL', 'INTC', 'CSCO', 'ADBE', 'ORCL'
]
```

```
def get_latest_timestamp(symbol):
    client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
    query = f"""
        SELECT MAX(timestamp) AS latest_timestamp
        FROM `{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}`
        WHERE symbol = '{symbol}'
    """
    query_job = client.query(query)
    result = query_job.result()

    latest_timestamp_row = list(result)
    if latest_timestamp_row:
        return latest_timestamp_row[0].latest_timestamp
    else:
        return None
```

```
def fetch_and_store_data(symbol, start_date, end_date):
    endpoint = f'{BASE_URL}/v2/stocks/{symbol}/bars'
    params = {
        'start': start_date,
        'end': end_date,
        'timeframe': '1Day',
        'adjustment': 'split',
        'feed': 'sip',
        'limit': 10000
    }
```

```
client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
table_id = f'{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}'
```

```
while True:
    response = requests.get(endpoint, headers=HEADERS, params=params)
    data = response.json()

    if 'bars' in data:
```

```
rows_to_insert = []
```

```
utc_zone = pytz.utc
```

```
est_zone = pytz.timezone('America/New_York')
```

```
for bar in data['bars']:
```

```
    utc_timestamp = datetime.fromisoformat(bar['t'].replace('Z', '+00:00'))
```

```
    est_timestamp = utc_timestamp.astimezone(est_zone)
```

```
    rows_to_insert.append({
```

```
        'symbol': symbol,
```

```
        'close_price': bar['c'],
```

```
        'high_price': bar['h'],
```

```
        'low_price': bar['l'],
```

```
        'number_of_trades': bar['n'],
```

```
        'open_price': bar['o'],
```

```
        'timestamp': est_timestamp.isoformat(),
```

```
        'volume': bar['v'],
```

```
        'volume_weighted_average_price': bar['vw']
```

```
    })
```

```
errors = client.insert_rows_json(table_id, rows_to_insert)
```

```
if errors:
```

```
    print(f"Errors while inserting data for {symbol}: {errors}")
```

```
else:
```

```
    print(f"Data for {symbol} successfully stored in BigQuery.")
```

```
else:
```

```
    print(f"No data fetched for {symbol}.")
```

```
next_page_token = data.get('next_page_token')
```

```
if next_page_token:
```

```
    params['page_token'] = next_page_token
```

```
else:
```

```
    break
```

```
def run_etl():
```

```
    end_date = datetime.now(timezone.utc) - timedelta(minutes=15) # Fetch data until yesterday
```

```
    start_date = end_date - timedelta(days=2500) # Fetch data for the last 7 years
```

```
    for symbol in TOP_20_STOCKS:
```

```
        latest_timestamp = get_latest_timestamp(symbol)
```

```
        if latest_timestamp:
```

```
            # Adjust start_date to just after the latest timestamp
```

```
            start_date = max(latest_timestamp + timedelta(days=1), end_date - timedelta(days=2500))
```

```

else:
    # Use the original start_date if no data exists
    start_date = end_date - timedelta(days=2500)

    start_date_str = start_date.strftime('%Y-%m-%dT%H:%M:%SZ')
    end_date_str = end_date.strftime('%Y-%m-%dT%H:%M:%SZ')

    fetch_and_store_data(symbol, start_date_str, end_date_str)

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'historical_stock_data_1_day_interval',
    default_args=default_args,
    description='DAG to load historical stock data into BigQuery',
    schedule_interval=None, # Adjust frequency if needed
    catchup=False,
)

etl_task = PythonOperator(
    task_id='run_etl',
    python_callable=run_etl,
    dag=dag,
)

etl_task

```

### **Pulls current data with 1day interval (run every day)**

```

from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta, timezone
import requests
from google.cloud import bigquery

```

```
from google.oauth2 import service_account
import pytz
```

```
# Configuration
```

```
ALPACA_API_KEY = 'PKAFPE8IL5495B6P4106'
ALPACA_API_SECRET = 'yde2oINHNXvDSUpsJUKART2pWJ7sMeXEbz2Oe4vp'
BASE_URL = 'https://data.alpaca.markets'
HEADERS = {'APCA-API-KEY-ID': ALPACA_API_KEY, 'APCA-API-SECRET-KEY': ALPACA_API_SECRET}
BIGQUERY_PROJECT_ID = 'stocksdatacsv-433003'
BIGQUERY_DATASET = 'stocks_data'
BIGQUERY_TABLE = 'historical_stock_prices'
# METADATA_TABLE = 'etl_metadata'
SERVICE_ACCOUNT_FILE = '/opt/airflow/dags/stocksdatacsv-433003-f8dc42626436.json'
```

```
# List of top 20 US stocks
```

```
TOP_20_STOCKS = [
    'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'META', 'NVDA', 'BRK.B', 'JPM', 'V',
    'MA', 'WMT', 'DIS', 'HD', 'NFLX', 'PYPL', 'INTC', 'CSCO', 'ADBE', 'ORCL'
]
```

```
def get_latest_timestamp(symbol):
```

```
    client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
    query = f"""
        SELECT MAX(timestamp) AS latest_timestamp
        FROM `{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}`
        WHERE symbol = '{symbol}'
    """
```

```
    query_job = client.query(query)
    result = query_job.result()
```

```
    latest_timestamp_row = list(result)
    if latest_timestamp_row:
        return latest_timestamp_row[0].latest_timestamp
    else:
        return None
```

```
def fetch_and_store_data(symbol, start_date, end_date):
```

```
    endpoint = f'{BASE_URL}/v2/stocks/{symbol}/bars'
    params = {
        'start': start_date,
        'end': end_date,
        'timeframe': '1Day', # Fetch data with a 1-day timeframe
        'adjustment': 'split',
        'feed': 'sip',
    }
```



```
'limit': 10000
}
```

```
client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
table_id = f'{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}'
```

```
while True:
```

```
    response = requests.get(endpoint, headers=HEADERS, params=params)
    data = response.json()
```

```
    if 'bars' in data:
```

```
        rows_to_insert = []
```

```
        # Define timezones
```

```
        utc_zone = pytz.utc
```

```
        est_zone = pytz.timezone('America/New_York')
```

```
        for bar in data['bars']:
```

```
            # Parse ISO 8601 timestamp
```

```
            utc_timestamp = datetime.fromisoformat(bar['t'].replace('Z', '+00:00'))
```

```
            est_timestamp = utc_timestamp.astimezone(est_zone)
```

```
            rows_to_insert.append({
```

```
                'symbol': symbol,
```

```
                'close_price': bar['c'],
```

```
                'high_price': bar['h'],
```

```
                'low_price': bar['l'],
```

```
                'number_of_trades': bar['n'],
```

```
                'open_price': bar['o'],
```

```
                'timestamp': est_timestamp.isoformat(), # Convert to ISO format for BigQuery
```

```
                'volume': bar['v'],
```

```
                'volume_weighted_average_price': bar['vw']
```

```
            })
```

```
    errors = client.insert_rows_json(table_id, rows_to_insert)
```

```
    if errors:
```

```
        print(f"Errors while inserting data for {symbol}: {errors}")
```

```
    else:
```

```
        print(f"Data for {symbol} successfully stored in BigQuery.")
```

```
else:
```

```
    print(f"No data fetched for {symbol}.")
```

```
# Check for pagination
```

```
next_page_token = data.get('next_page_token')
```

```

if next_page_token:
    params['page_token'] = next_page_token
else:
    break

```

```

def run_etl():
    end_date = datetime.now(timezone.utc) - timedelta(minutes=15) # Fetch data until yesterday
    start_date = end_date - timedelta(days=3) # Fetch data for the last day

```

```

for symbol in TOP_20_STOCKS:
    latest_timestamp = get_latest_timestamp(symbol)
    if latest_timestamp:
        # Ensure the start_date is just after the latest timestamp
        start_date = max(latest_timestamp + timedelta(days=1), end_date - timedelta(days=1))
    start_date_str = start_date.strftime('%Y-%m-%dT%H:%M:%SZ')
    end_date_str = end_date.strftime('%Y-%m-%dT%H:%M:%SZ')

```

```

    fetch_and_store_data(symbol, start_date_str, end_date_str)

```

```

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

```

```

dag = DAG(
    'daily_stock_data_load',
    default_args=default_args,
    description='DAG to load daily stock data into BigQuery',
    schedule_interval='@daily', # Run once daily
    catchup=False,
)

```

```

etl_task = PythonOperator(
    task_id='run_etl',
    python_callable=run_etl,
    dag=dag,
)

```

```

etl_task

```

## Historic data load, Python dag to pull data with 5 mins interval (Run once)

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta, timezone
import requests
from google.cloud import bigquery
from google.oauth2 import service_account
import pytz

# Configuration
ALPACA_API_KEY = 'PKAFPE8IL5495B6P4106'
ALPACA_API_SECRET = 'yde2oINHNXvDSUpsJUKART2pWJ7sMeXEbz2Oe4vp'
BASE_URL = 'https://data.alpaca.markets'
HEADERS = {'APCA-API-KEY-ID': ALPACA_API_KEY, 'APCA-API-SECRET-KEY': ALPACA_API_SECRET}
SERVICE_ACCOUNT_FILE = '/opt/airflow/dags/stocksdatacsv-433003-f8dc42626436.json'
BIGQUERY_PROJECT_ID = 'stocksdatacsv-433003'
BIGQUERY_DATASET = 'stocks_data'
BIGQUERY_TABLE = 'realtime_stock_prices'

# List of top 20 US stocks
TOP_20_STOCKS = [
    'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'META', 'NVDA', 'BRK.B', 'JPM', 'V',
    'MA', 'WMT', 'DIS', 'HD', 'NFLX', 'PYPL', 'INTC', 'CSCO', 'ADBE', 'ORCL'
]

def fetch_and_store_data(symbol, start_date, end_date):
    endpoint = f'{BASE_URL}/v2/stocks/{symbol}/bars'
    params = {
        'start': start_date,
        'end': end_date,
        'timeframe': '5Min',
        'adjustment': 'split',
        'feed': 'sip',
        'limit': 10000
    }

    client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
    table_id = f'{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}'
```

```

while True:
    response = requests.get(endpoint, headers=HEADERS, params=params)
    data = response.json()

    if 'bars' in data:
        rows_to_insert = []

        # Define timezones
        utc_zone = pytz.utc
        est_zone = pytz.timezone('America/New_York')

        for bar in data['bars']:
            # Parse ISO 8601 timestamp
            utc_timestamp = datetime.fromisoformat(bar['t'].replace('Z', '+00:00'))
            est_timestamp = utc_timestamp.astimezone(est_zone)

            rows_to_insert.append({
                'symbol': symbol,
                'close_price': bar['c'],
                'high_price': bar['h'],
                'low_price': bar['l'],
                'number_of_trades': bar['n'],
                'open_price': bar['o'],
                'timestamp': est_timestamp.isoformat(), # Convert to ISO format for BigQuery
                'volume': bar['v'],
                'volume_weighted_average_price': bar['vw']
            })

        errors = client.insert_rows_json(table_id, rows_to_insert)
        if errors:
            print(f"Errors while inserting data for {symbol}: {errors}")
        else:
            print(f"Data for {symbol} successfully stored in BigQuery.")
    else:
        print(f"No data fetched for {symbol}.")

    # Check for pagination
    next_page_token = data.get('next_page_token')
    if next_page_token:
        params['page_token'] = next_page_token
    else:
        break

```

```

def run_etl():
    end_date = datetime.now(timezone.utc) - timedelta(minutes=15)
    start_date = end_date - timedelta(days=1080) # Almost 3 year
    interval = timedelta(days=90) # 3 months

    while start_date < end_date:
        interval_end_date = start_date + interval

        if interval_end_date > end_date:
            interval_end_date = end_date

        start_date_str = start_date.strftime('%Y-%m-%dT%H:%M:%SZ')
        interval_end_date_str = interval_end_date.strftime('%Y-%m-%dT%H:%M:%SZ')

        for symbol in TOP_20_STOCKS:
            fetch_and_store_data(symbol, start_date_str, interval_end_date_str)

        start_date += interval # Move to the next 3-month interval

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'historic_stock_data_load',
    default_args=default_args,
    description='One-time DAG to load historic stock data into BigQuery',
    schedule_interval=None,
    catchup=False,
)

etl_task = PythonOperator(
    task_id='run_etl',
    python_callable=run_etl,
    dag=dag,
)

etl_task

```

### **Pull current data with 5 mins interval (runs every 10 mins)**

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta, timezone
import requests
from google.cloud import bigquery
from google.oauth2 import service_account
import pytz

# Configuration
ALPACA_API_KEY = 'PKAFPE8IL5495B6P4106'
ALPACA_API_SECRET = 'yde2oINHNXvDSUpsJUKART2pWJ7sMeXEbz2Oe4vp'
BASE_URL = 'https://data.alpaca.markets'
HEADERS = {'APCA-API-KEY-ID': ALPACA_API_KEY, 'APCA-API-SECRET-KEY': ALPACA_API_SECRET}
SERVICE_ACCOUNT_FILE = '/opt/airflow/dags/stocksdatacsv-433003-f8dc42626436.json'
BIGQUERY_PROJECT_ID = 'stocksdatacsv-433003'
BIGQUERY_DATASET = 'stocks_data'
BIGQUERY_TABLE = 'realtime_stock_prices'

# List of top 20 US stocks
TOP_20_STOCKS = [
    'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'META', 'NVDA', 'BRK.B', 'JPM', 'V',
    'MA', 'WMT', 'DIS', 'HD', 'NFLX', 'PYPL', 'INTC', 'CSCO', 'ADBE', 'ORCL'
]

def get_latest_timestamp(symbol):
    client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
    query = f"""
        SELECT MAX(timestamp) AS latest_timestamp
        FROM `{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}`
        WHERE symbol = '{symbol}'
    """
    query_job = client.query(query)
    result = query_job.result()

    latest_timestamp_row = list(result)
    if latest_timestamp_row:
        return latest_timestamp_row[0].latest_timestamp
    else:
        return None

def fetch_and_store_data(symbol, start_date, end_date):
```

```

endpoint = f'{BASE_URL}/v2/stocks/{symbol}/bars'
params = {
    'start': start_date,
    'end': end_date,
    'timeframe': '5Min',
    'adjustment': 'split',
    'feed': 'sip',
    'limit': 10000
}

client = bigquery.Client.from_service_account_json(SERVICE_ACCOUNT_FILE)
table_id = f'{BIGQUERY_PROJECT_ID}.{BIGQUERY_DATASET}.{BIGQUERY_TABLE}'

while True:
    response = requests.get(endpoint, headers=HEADERS, params=params)
    data = response.json()

    if 'bars' in data:
        rows_to_insert = []

        # Define timezones
        utc_zone = pytz.utc
        est_zone = pytz.timezone('America/New_York')

        for bar in data['bars']:
            # Parse ISO 8601 timestamp
            utc_timestamp = datetime.fromisoformat(bar['t'].replace('Z', '+00:00'))
            est_timestamp = utc_timestamp.astimezone(est_zone)

            rows_to_insert.append({
                'symbol': symbol,
                'close_price': bar['c'],
                'high_price': bar['h'],
                'low_price': bar['l'],
                'number_of_trades': bar['n'],
                'open_price': bar['o'],
                'timestamp': est_timestamp.isoformat(), # Convert to ISO format for BigQuery
                'volume': bar['v'],
                'volume_weighted_average_price': bar['vw']
            })

        errors = client.insert_rows_json(table_id, rows_to_insert)
        if errors:
            print(f"Errors while inserting data for {symbol}: {errors}")

```

```

        else:
            print(f"Data for {symbol} successfully stored in BigQuery.")
    else:
        print(f"No data fetched for {symbol}. Continuing to next symbol.")

    # Check for pagination
    next_page_token = data.get('next_page_token')
    if next_page_token:
        params['page_token'] = next_page_token
    else:
        break

def run_etl():
    end_date = datetime.now(timezone.utc) - timedelta(minutes=15) # To avoid overlap with the current
    minute
    start_date = end_date - timedelta(days=7) # Time window to fetch data

    for symbol in TOP_20_STOCKS:
        latest_timestamp = get_latest_timestamp(symbol)
        if latest_timestamp:
            # Ensure the start_date is just after the latest timestamp
            start_date = max(latest_timestamp + timedelta(minutes=1), end_date - timedelta(days=7))
            start_date_str = start_date.strftime('%Y-%m-%dT%H:%M:%SZ')
            end_date_str = end_date.strftime('%Y-%m-%dT%H:%M:%SZ')

            fetch_and_store_data(symbol, start_date_str, end_date_str)

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=3),
}

dag = DAG(
    'realtime_stock_data_load',
    default_args=default_args,
    description='DAG to load real-time stock data into BigQuery every 10 minutes',
    schedule_interval='*/10 * * * *', # Run every 10 minutes
    catchup=False,
)

```



```
etl_task = PythonOperator(  
    task_id='run_etl',  
    python_callable=run_etl,  
    dag=dag,  
)
```

etl\_task

#### **FYI: Common Docker commands:**

**Remove all containers:** docker container rm -f \$(docker container ls -aq)

**Remove all images:** docker image rm -f \$(docker image ls -q)

**Run docker compose file in detached mode:** docker-compose up -d

**Run docker compose with debug:** docker-compose up --build

**Stop docker compose:** docker-compose down

**Check docker volumes:** docker volume ls

**To delete all docker volumes:** docker volume rm \$(docker volume ls -q)

**See running containers:** docker ps

**NOTE:** To remove airflow metadata or the logs from the Postgres DB, you need to run these commands:

- psql -h postgres -U airflow -d airflow
- \dt
- SELECT \* FROM dag\_run LIMIT 10;
- DELETE FROM dag\_run WHERE execution\_date < NOW() - INTERVAL '2 days';
- VACUUM FULL dag\_run;